Data Scientist Capstone Project

Soumaya Jameleddine

June 2019

Redefining Cancer Treatment

Project overview

Genetic testing has been gaining a lot of traction in the past couple of years and we can expect it to disrupt the way we treat various diseases like cancer once we fully harness the power of data and machine learning. This project is an old Kaggle competition that falls in that category and was launched by Memorial Sloan Kettering Cancer Center (MSKCC).

Project Statement

A cancer tumor can have thousands of genetic mutations. The challenge is differentiating between mutations that contribute to the tumor growth (drivers) from the neutral mutations (passengers). Currently this work is heavily relying on manual intervention where clinical pathologist review and classify every single genetic mutation based on evidence from text_based clinical literature. This task is very time consuming and can be a good candidate for modeling thanks to more advanced and sophisticated NLP algorithms.

For this competition MSKCC provided an expert-annotated knowledge base where world-class researchers and oncologists have manually annotated thousands of mutations.

Scoring (Metrics)

This project is a classification problem where we are tasked to predict a mutation class. There are 9 different classes and Kaggle/MSKCC didn't provide their definition which makes finetuning and customizing the scoring a difficult task. In a typical classification problem, we can use different scoring metrics: accuracy, f1-scores (macro for multiclasses) and f-b scores if we want to overweight either recall or precision. In this case since we don't know what classes represent driver mutations vs passenger mutations, so we have to forgo the f-b scores. The data is also unbalanced and therefore we cannot rely on accuracy alone but we need to look at f1-scores and confusion matrices when performing model selection.

I. Analysis

1. Data Exploration and Visualization

After merging the two training tables provided by MSKCC we get a table with 5 fields:

- ID: identification of the case/patient
- Text: the clinically annotated text describing the case and the mutation
- Gene
- Variation (which type of variable did the gene undergo)
- Class (Target)

I first started by looking at the Class distribution:

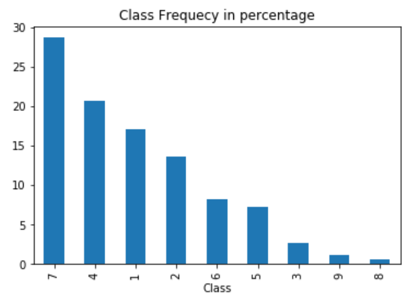


Figure1: Class Frequency

As mentioned in the scoring section, the data is rather ***unbalanced*** so I will have to correct for this during our training.

~28% of the cases fall under class 7 followed by ~20% in class 4, only a small percentage fall in class 8/9.

Next, I looked at the genes intervening in the different classes. The plot below shows the top 10 genes for each class.

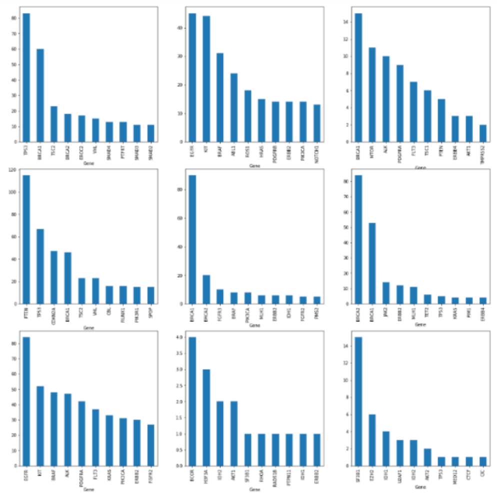
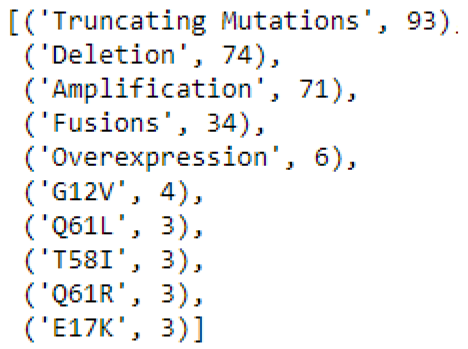


Figure2: top 10 genes for each class

I can make several observations:

- There are some overlap in the top 10 genes between different classes (BRCA1, BRCA2, TP53…)
- Some classes have dominant genes (Class 5 in the center) vs others where the distribution is a bit more uniform (Class 3 top right)
- Genes are important features, but will need extra feature engineering

Now it is time to look at the variations. I started by looking at the most common variations.



```
[('Truncating Mutations', 93),
 ('Deletion', 74),
 ('Amplification', 71),
 ('Fusions', 34),
 ('Overexpression', 6),
 ('G12V', 4),
 ('Q61L', 3),
 ('T58I', 3),
 ('Q61R', 3),
 ('E17K', 3)]
```

I can see the variations can take different forms, some don't have a gene specification in them but only an operation like truncating mutations, deletion, Amplification, fusion, overexpression and other have what seems to be a first letter of a gene a location and another gene identification which after several research points to a substitution operation. I will continue working on variation manipulation in the data pre-processing and feature engineering section.

2. Algorithms and Techniques

The data presented has different features: some categorical, some numerical and other textual. Therefore, I needed to think about this project as a combination of NLP algorithms and a classifier.

On the NLP side I used different methodologies from TF-IDF to Gensim Doc2Vec.

TF-IDF refers to term frequency -inverse document frequency. The objective of the TF-IDF is to identify the topic (specificity of a document) by detecting relevant words that are not common across the board. TF-IDF doesn't take into account the context or the surrounding of a particular words. It is also a pure numeric statistic.

Word2Vec and Doc2Vec by extension are product od neural network trained to reconstruct the context of words. Word2Vec and doc2Vec will capture similarities and relationships between certain words. In this case and as a non-health care professional and not familiar with the corpus, I see a case to experiment with both methodologies.

On the classification side I used different classifiers from logistic Regression to RandomForest, XGBClassifier and finally a Neural Network powered by Keras.

3. Benchmark

For this project I used two benchmarks to evaluate the performance of the model.
Since the data is unbalanced I can consider a dumb model that predicts everything to be in class 7 (dominant class) which sets the minimum accuracy to around 28%. I can also use a simple model like the combination of TF-IDF and logistic regression as another benchmark as it is common practice to choose a linear model when benchmarking.

II. Methodology

1. Data Pre-processing and feature engineering

<u>Variation feature engineering:</u>
As mentioned in the previous section, I noticed that the variation field contains different information among them the type of operation and/or gene(s) involved and the location of the mutation.
For this purpose, I needed to perform a couple of manipulations using the regEx package to extract as many details as possible.

These are the different types of operations I mapped and the potential annotations it can take.
truncations = ['truncating mutations','trunc']
deletion_insertion = ['insertions/deletions','deletion/insertion','delins']
fusion = ['fusions','fusion','fus','fs','fs']*
deletion = ['deletions','deletion','del']
insertion = ['insertions','insertion','ins']
amplification = ['amplification']
overexpression= ['overexpression']
splice= ['splice']
duplication= ['duplications','duplication','dup']

I added a feature *operation* where I inserted the operation based on the above-mentioned annotation. The other major (and common) operation is substitution which doesn't come with a tag but has a uniform format with letter/location/letter. For that matter I identified variations in the format, tagged the operation as substitution but also extracted the *first letter*, *the location* and the *last letter*.

The last operation performed on variation was for fusion where I noticed the original gene was present along with a *second gene for a particular format*. Therefore, I added a second gene field.

I looked at the presence of the different variations across classes.

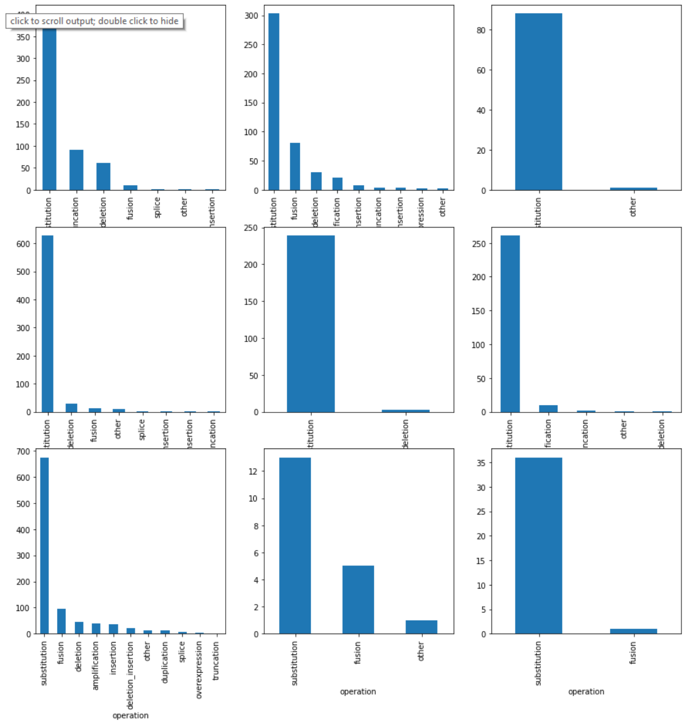


Figure3: Distribution of operations for each class

Substitution is by far the most common operation. For that reason, I hope the next layer (gene involved and location) will help differentiate between classes. Besides, some operations are only present for a couple of classes.

Text processing:

In order to use the text, I started by cleaning the data by removing special characters, stop words and common words in the medical field like result method figure and table. I used both regex and NLTK to do these basic texts manipulations, remove some special characters, use lower case, lemmatizing. These are the 10most common words in the corpus: no big surprise here.

	Word	Frequency
0	mutation	341366
1	cell	300201
2	protein	121465
3	mutant	120750
4	tumor	115706
5	cancer	113104
6	gene	100578
7	variant	89851
8	patient	86436
9	activity	83770
10	domain	82923
11	kinase	70896
12	assay	57454
13	brca1	57402
14	line	51183

I also looked at the distribution of number of words per class to see whether the length of the text can be a signal of how serious the case is.

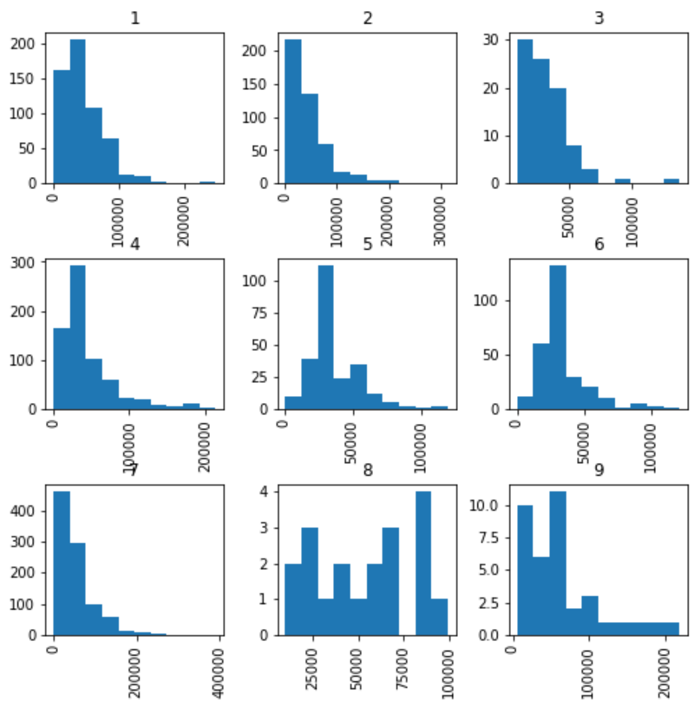


Figure4: Length of text for each class

We notice that some classes have fatter tails than others so we will add this as a feature.

2. **Implementation**

   a. <u>With TF-IDF:</u>

In order to assemble textual and non-textual features I recreated a fit-transform methods that will treat those two categories differently, then fed them in a pipeline to transform the features accordingly.

I used a selection of classifiers:

- Logistic Regression (that I picked as a benchmark)
- SVC
- RandomForestClassifier
- XGBClassifier
- MLP
- Neural Network

Since XGB was one of the classifiers, I added a truncatedSVD in the pipeline.
TruncatedSVD performs a linear dimensionality reduction by truncating the singular value decomposition.

   b. <u>With Doc2Vec:</u>

I experimented with Doc2Vec three models using Gensim

Distributed Bag of Words (DBOW):

The paragraph vectors are obtained by training a neural network to predict a probability distribution of words in a paragraph given a randomly-sampled word from the paragraph

Distributed Memory (DM)

Distributed Memory (DM) acts as a memory that remembers what is missing from the current context. The document vector intends to represent the concept of a document.

If dm=0 then the distributed bag of words (PV-DBOW) is used

If dm=1 then the distributed memory (PV-DM) is used

Here is how we instantiate the two models.

```
d2v_model_dm = gensim.models.Doc2Vec(min_count=3, window=7, size=400, sample=1e-3, negative=5, dm=1,workers=2)
d2v_model_dbow = gensim.models.Doc2Vec(min_count=3, window=7, size=400, sample=1e-3, negative=5, dm=0, workers=2)
```

We also added negative sampling (negative=5) in order computation burden by updating a small percentage of the model's weights.

3. **Refinement:**

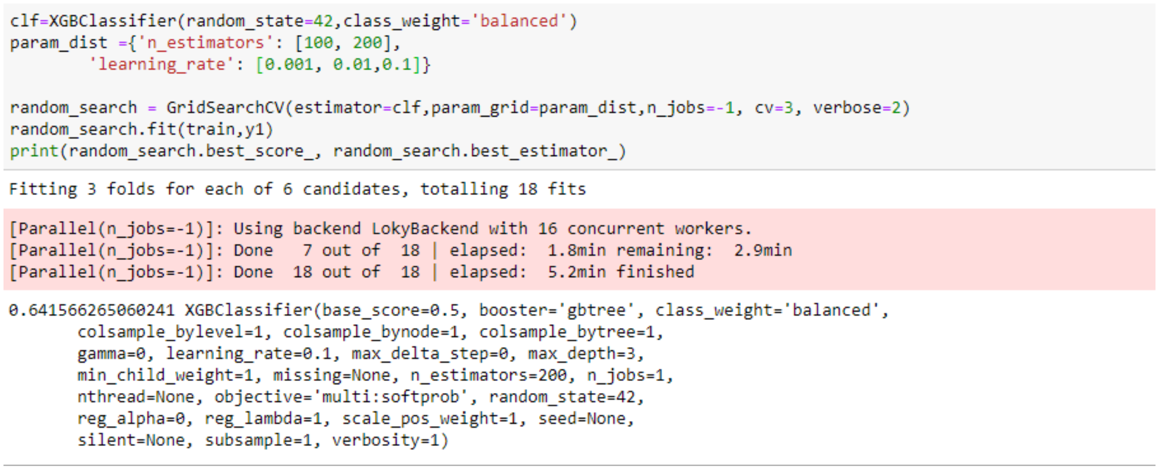
The refinement process relied heavily on tuning the models' hyperparameters.

For the XGB Classifier we tuned the:

- ❖ Class_weight: to set it to “balanced” to take into account the imbalanced nature of our data.

  We also used a gridsearch to fine tune
- ❖ n_estimators
- ❖ min_sample_split
- ❖ learning_rate

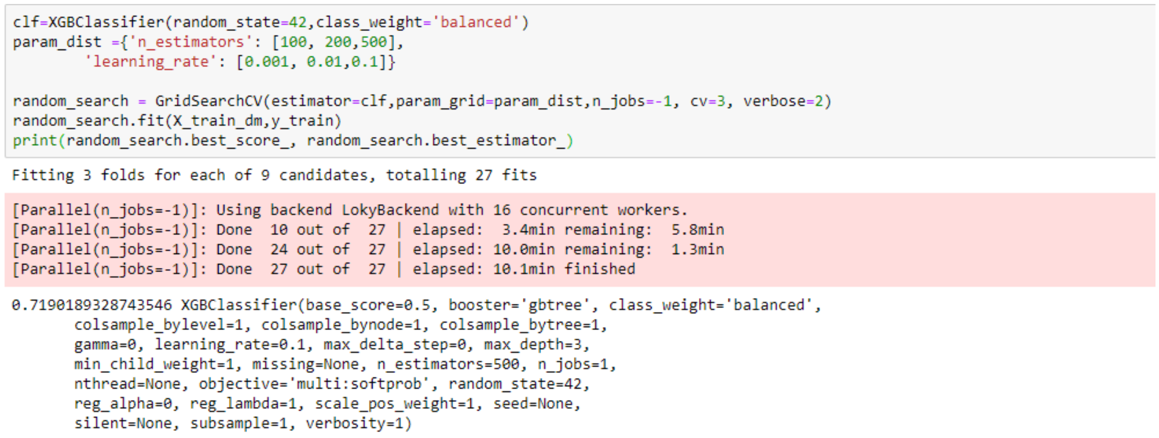
<u>GridSearch for XGB using TF-IDF transformations:</u>



```
clf=XGBClassifier(random_state=42,class_weight='balanced')
param_dist ={'n_estimators': [100, 200],
        'learning_rate': [0.001, 0.01,0.1]}

random_search = GridSearchCV(estimator=clf,param_grid=param_dist,n_jobs=-1, cv=3, verbose=2)
random_search.fit(train,y1)
print(random_search.best_score_, random_search.best_estimator_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   7 out of  18 | elapsed:  1.8min remaining:  2.9min
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:  5.2min finished

0.641566265060241 XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=200, n_jobs=1,
       nthread=None, objective='multi:softprob', random_state=42,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
```

<u>GridSearch for XGB using doc2Vec transformations</u>



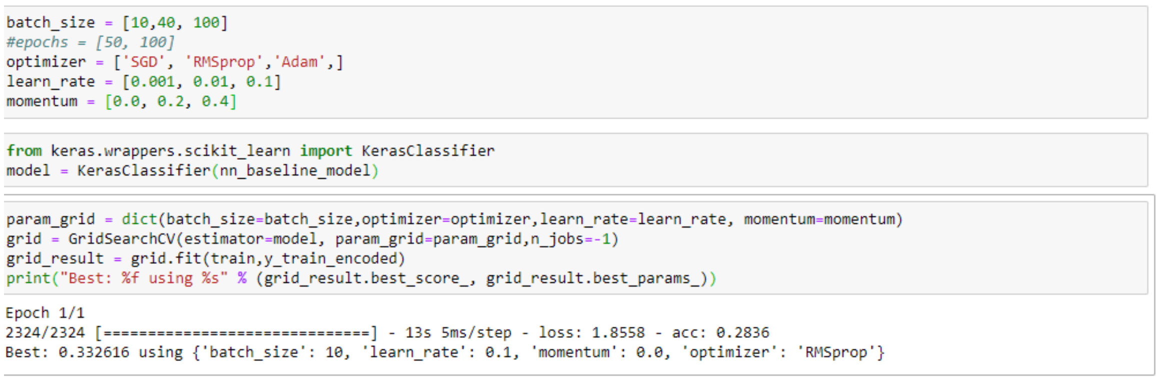
```
clf=XGBClassifier(random_state=42,class_weight='balanced')
param_dist ={'n_estimators': [100, 200,500],
        'learning_rate': [0.001, 0.01,0.1]}

random_search = GridSearchCV(estimator=clf,param_grid=param_dist,n_jobs=-1, cv=3, verbose=2)
random_search.fit(X_train_dm,y_train)
print(random_search.best_score_, random_search.best_estimator_)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  27 | elapsed:  3.4min remaining:  5.8min
[Parallel(n_jobs=-1)]: Done  24 out of  27 | elapsed: 10.0min remaining:  1.3min
[Parallel(n_jobs=-1)]: Done  27 out of  27 | elapsed: 10.1min finished

0.7190189328743546 XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=500, n_jobs=1,
       nthread=None, objective='multi:softprob', random_state=42,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
```

For the neural network, we also used a GridSearch to fine tune

- ❖ batch size
- ❖ optimizer
- ❖ learn rate
- ❖ momentum

<u>Grid Search on NN (combined with TF-IDF)</u>



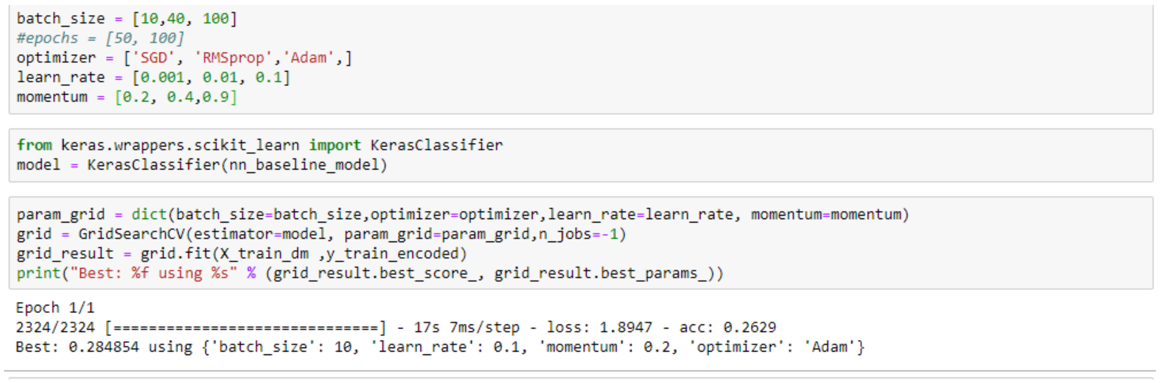
```
batch_size = [10,40, 100]
#epochs = [50, 100]
optimizer = ['SGD', 'RMSprop','Adam',]
learn_rate = [0.001, 0.01, 0.1]
momentum = [0.0, 0.2, 0.4]
```

```
from keras.wrappers.scikit_learn import KerasClassifier
model = KerasClassifier(nn_baseline_model)
```

```
param_grid = dict(batch_size=batch_size,optimizer=optimizer,learn_rate=learn_rate, momentum=momentum)
grid = GridSearchCV(estimator=model, param_grid=param_grid,n_jobs=-1)
grid_result = grid.fit(train,y_train_encoded)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Epoch 1/1
2324/2324 [==============================] - 13s 5ms/step - loss: 1.8558 - acc: 0.2836
Best: 0.332616 using {'batch_size': 10, 'learn_rate': 0.1, 'momentum': 0.0, 'optimizer': 'RMSprop'}
```

Grid Search on NN (combined with Doc2Vec)



```
batch_size = [10,40, 100]
#epochs = [50, 100]
optimizer = ['SGD', 'RMSprop','Adam',]
learn_rate = [0.001, 0.01, 0.1]
momentum = [0.2, 0.4,0.9]
```

```
from keras.wrappers.scikit_learn import KerasClassifier
model = KerasClassifier(nn_baseline_model)
```

```
param_grid = dict(batch_size=batch_size,optimizer=optimizer,learn_rate=learn_rate, momentum=momentum)
grid = GridSearchCV(estimator=model, param_grid=param_grid,n_jobs=-1)
grid_result = grid.fit(X_train_dm ,y_train_encoded)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Epoch 1/1
2324/2324 [==============================] - 17s 7ms/step - loss: 1.8947 - acc: 0.2629
Best: 0.284854 using {'batch_size': 10, 'learn_rate': 0.1, 'momentum': 0.2, 'optimizer': 'Adam'}
```

III. Results

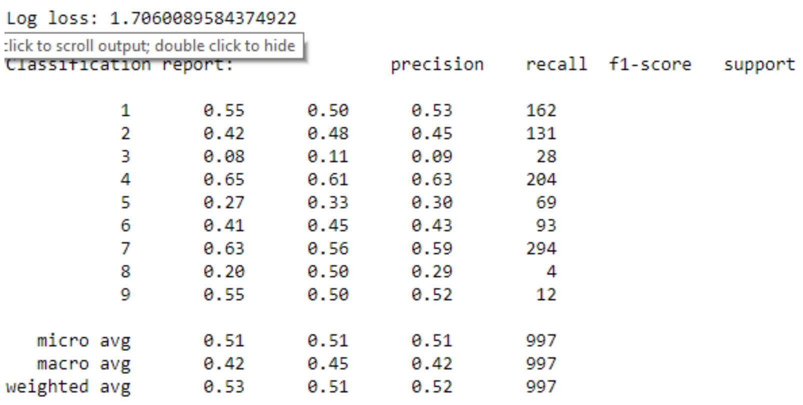
1. Model Validation and Results

Here is a summary of the different models used:

NLP	Classifiers	Accuracy	Macro avg F1
TF-IDF	Logistic Regression	0.52	0.35
	SVC	0.33	0.12
	RandomForest	0.53	0.39
	XGB	0.60	0.47
	MLP	0.54	0.35
	Neutral Network	0.52	
d2v_model_dm	XGB	0.62	0.49
d2v_model_dbow	XGB	0.62	0.49
	NN	0.29	
TF-IDF combined with Logistic Regression was chosen as my benchmark			

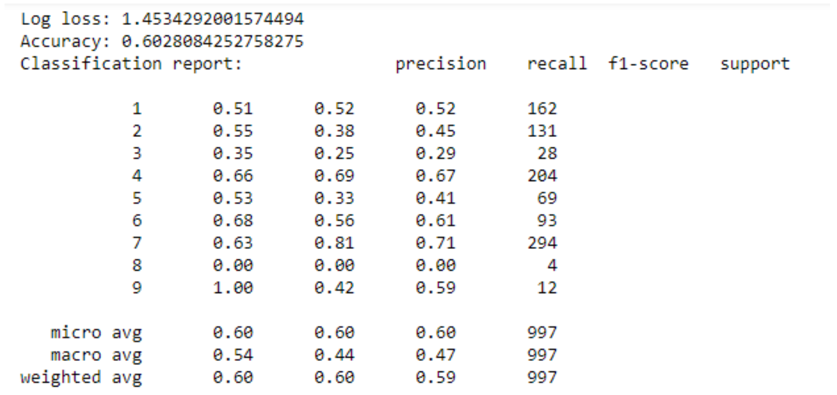
XGBClassifier outperformed the rest of the classifiers. Its combination with TF-IDF and doc2Vec produced very similar results. Let's look more in depth at the classification report and the confusion report for both these models and see if this performance was uniform across the different classes.

Benchmark Classification report:



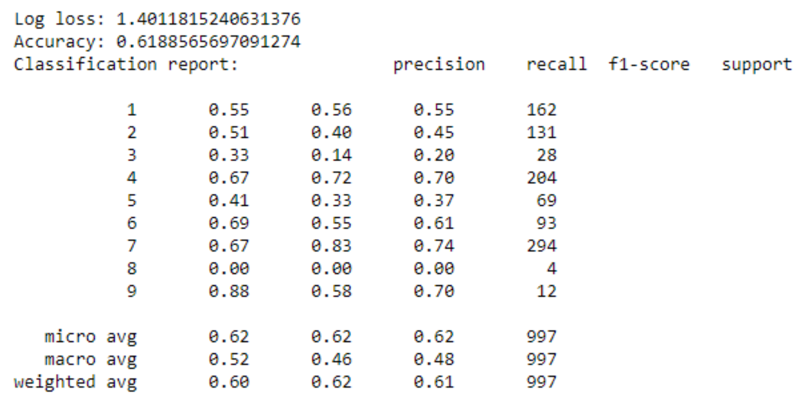
```
Log loss: 1.7060089584374922
:lick to scroll output; double click to hide
Classification report:                precision    recall  f1-score   support

           1       0.55      0.50      0.53       162
           2       0.42      0.48      0.45       131
           3       0.08      0.11      0.09        28
           4       0.65      0.61      0.63       204
           5       0.27      0.33      0.30        69
           6       0.41      0.45      0.43        93
           7       0.63      0.56      0.59       294
           8       0.20      0.50      0.29         4
           9       0.55      0.50      0.52        12

   micro avg       0.51      0.51      0.51       997
   macro avg       0.42      0.45      0.42       997
weighted avg       0.53      0.51      0.52       997
```

TF-IDF + XGB Classification report:



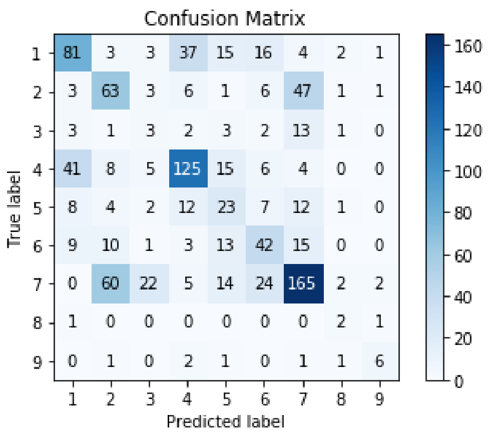
```
Log loss: 1.4534292001574494
Accuracy: 0.6028084252758275
Classification report:                precision    recall  f1-score   support

           1       0.51      0.52      0.52       162
           2       0.55      0.38      0.45       131
           3       0.35      0.25      0.29        28
           4       0.66      0.69      0.67       204
           5       0.53      0.33      0.41        69
           6       0.68      0.56      0.61        93
           7       0.63      0.81      0.71       294
           8       0.00      0.00      0.00         4
           9       1.00      0.42      0.59        12

   micro avg       0.60      0.60      0.60       997
   macro avg       0.54      0.44      0.47       997
weighted avg       0.60      0.60      0.59       997
```

Doc2Vec + XGB Classification report:

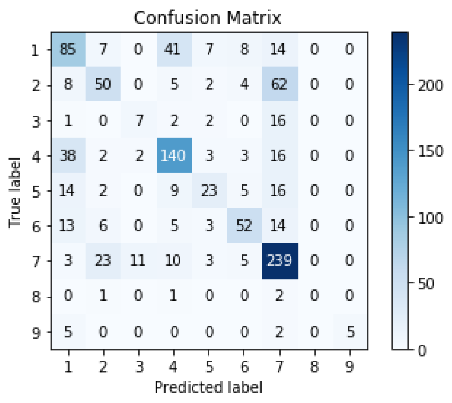


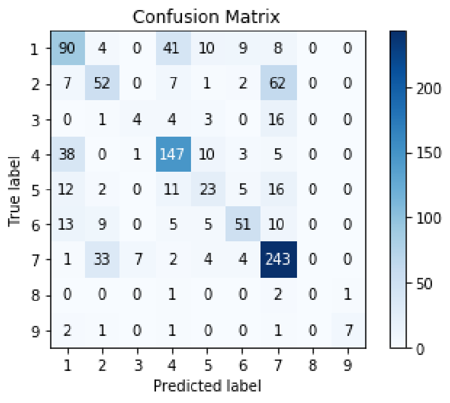
```
Log loss: 1.4011815240631376
Accuracy: 0.6188565697091274
Classification report:                precision    recall  f1-score   support

           1       0.55      0.56      0.55       162
           2       0.51      0.40      0.45       131
           3       0.33      0.14      0.20        28
           4       0.67      0.72      0.70       204
           5       0.41      0.33      0.37        69
           6       0.69      0.55      0.61        93
           7       0.67      0.83      0.74       294
           8       0.00      0.00      0.00         4
           9       0.88      0.58      0.70        12

   micro avg       0.62      0.62      0.62       997
   macro avg       0.52      0.46      0.48       997
weighted avg       0.60      0.62      0.61       997
```

TF-IDF+Logistic Regression



TF-IDF + XGB Confusion Matrix vs Doc2Vec + XGB Confusion Matrix





Both models have difficulties differentiating between class 1 and 4 and class 2 and 7. Other models analyzed and not discussed in this document have exhibited the same shortcoming. Overall the results are really close to each other. For that reason, I would lean to the TF-IDF combined with the XGB classifier as this model is simpler and easier to grasp. The philosophy being if results are similar we should pick the less complex model.

2. Reflection and Ideas for further improvement

Being an outsider to the health care domain I spent a lot of time researching mutations and trying to understand the data presented. The more I spent time on this project the more ideas I had on how to further improve it. I listed below a couple of items for each phase of the project.

For feature engineering:

Although the results were promising, I think we would benefit from using a medical dictionary that better captures and categorizes medical terms. We can also experiment with state-of-the-art NLP packages like Bert and Elmo. On the feature engineering side, partnering with an oncologist would have undoubtedly led to better features and therefore better results. We can work with the specialist to try to understand what makes class 1&4 and 2&7 similar and think about new features that can help us better distinguish between the two of them.

For modeling:
In this project I performed a model selection. Another approach, popular among the Kaggle community is model ensembles either by using a majority vote or by using a general model to ensemble the "sub-models".

For scoring:
In practice, knowing what the different classes represent is extremely valuable, we would rather flag a healthy patient for further screening than miss a patient with cancer. In these cases, errors can be lethal.

Conclusion

This project has been an exciting project to research and work on. Genetic diagnosis is the future of medicine and although the medical field made a lot of process in the recent years, the field is relatively in its infancy. As for the data scientist this is both a goldmine and challenge. Just think about all these combinations and think about the impact you can have!